Part A: System Documentation

WEEK Sep. 08 – 12

1. Requirements Elicitation

1.1. Functional Requirements

The Banking System shall provide the following functional capabilities:

1. Customer Management

- Register new customers with personal details (name, address, contact information)

- Update existing customer information

- View customer details and account relationships

2. Account Management

- Create new accounts of different types (Savings, Investment, Cheque)

- Enforce account-specific creation rules:

- Investment accounts require minimum BWP 500.00 initial deposit

- Cheque accounts require employment verification

- Support multiple accounts per customer

- Close accounts upon customer request

3. Transaction Processing

- Accept deposits to any account type

- Process withdrawals from eligible accounts (Investment and Cheque only)

- Maintain transaction history for all accounts

- Prevent invalid transactions (e.g., withdrawals from Savings accounts)

4. Interest Calculation

- Automatically calculate monthly interest for eligible accounts

- Apply appropriate interest rates:

- 5% monthly for Investment accounts

- 0.05% monthly for Savings accounts

- Credit interest amounts to account balances

5. Reporting

- Generate account statements

- Provide balance inquiries

- View transaction histories

1.2. Non-Functional Requirements

1. Security

- Protect sensitive customer financial data

- Implement input validation to prevent invalid transactions

- Ensure logical security constraints (e.g., no Savings account withdrawals)

2. Usability

- Intuitive interface for bank staff

- Clear error messages and transaction feedback

- Comprehensive help and documentation

3. Reliability

- 99.5% system availability during banking hours

- Accurate financial calculations

- Data consistency through transaction integrity

4. Performance

- Response time under 2 seconds for most operations

- Support for concurrent users

- Efficient interest calculation process

5. Maintainability

- Well-documented code following OOP principles

- Modular design for easy future enhancements

- Comprehensive logging for troubleshooting

Appendicle: Interview Record

Date of Interview| September 18, 2025 |

Time of Interview | 10:30AM – 12:12PM |

Interviewer(s) | Naomi Bontshitswe/ Cse24 group

Interviewee/Role | Mr. Kentsenao Baseki

Medium | Microsoft Teams Online Meeting |

2. Interview Questions & Answers

| 1 | Should monthly interest require human approval?

A: No, the process must be \*\*fully automated\*\*. The system should calculate based on the current balance and credit the amount, recording it in the transaction history.

2 . What critical customer information must be stored beyond basics? |

A: As the analyst, you must identify this. \*\*Common attributes\*\* (e.g., address, contact) should be inherited. \*\*Differentiating attributes\*\* include: for an Individual ,national ID, date of birth; for a Company- company number, date of incorporation.

|3 | What is a company's unique identifier?

A: A company registration number (e.g., "BW..."), which is different in data type and format from an individual's national ID number.

4. Should we handle incorrect login attempts (e.g., lock after 3 tries)?

A: No. The focus is on core OO design, not advanced security features. Adding such features is "scope creep" and not worth the marks allocated.

5 . Can customers transfer funds between accounts?

A: No.This feature is not specified in the requirements and constitutes scope creep.

| 6 | Does the system support multiple concurrent users? |

A:\*\*Not a primary concern.\*\* This is a performance issue, and the focus is on the application's structure, not its scalability for this assignment. |

| 7 | Who is the primary user? Is it online banking? |

A:The bank teller is the primary user for account management functions (opening accounts). The customer is a user for viewing balances and transacting. The medium (online or in-bank) is not specified; focus on the functionality. |

| 8 | Are there eligibility requirements (e.g., age) for accounts? |

A :No. Do not overcomplicate. The goal is to model the core system, not real-world banking regulations. |

| 9 | What should happen if a transaction fails? |

A:An error message must be displayed (e.g., for insufficient funds). A failed transaction should not be recorded in the transaction history. |

| 10 | Does the system generate reports? |

A:The transaction history/account statement is the primary report. Do not implement additional reports. |

| 11 | Can a customer close an account? |

A:No.This is an example of scope creep. Stick to the outlined requirements: opening accounts, deposits, withdrawals, and viewing history. |

| 12 | Should interest calculation be polymorphic? |

A:Yes. This is a perfect scenario for applying polymorphism (e.g., through an interface) where each account type provides its own implementation of a `calculateInterest()` method. |

3. Key Requirements Elicited (Summary)

\* System must manage Individual and Company customers.

\* Three account types with specific minimum balances and interest rates.

\* Interest application is automatic and recorded.

\* A customer can hold multiple \*types\* of accounts.

\* \* Bank teller opens accounts.

\* Customers can deposit, withdraw, and view transaction history.

\* Full transaction history must be maintained.

\* GUI must be user-friendly and modern.

\* Focus is strictly on correct OO design and UML modeling, not on adding extra features.

2. Structural UML Modelling

2.1. System Use Case Diagram

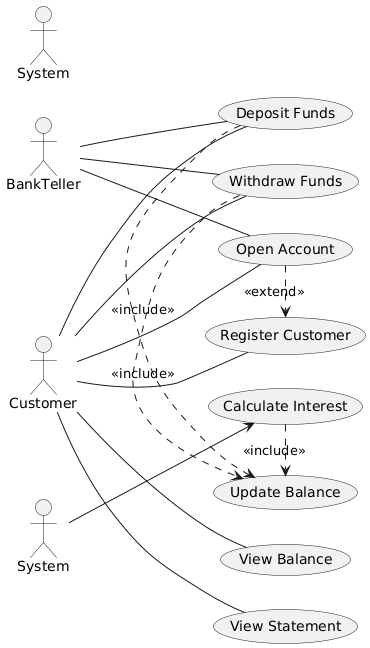


Figure 1: Banking System Use Case Diagram

Actors:

- Customer: Primary user who performs banking operations

- System: Automated actor that handles interest calculations

Use Cases:

- Register Customer

- Open Account (extends Register Customer)

- Deposit Funds (includes Update Balance)

- Withdraw Funds (includes Update Balance)

- View Balance

- View Statement

- Calculate Interest (includes Update Balance)

Relationships:

- «extends» relationship between Open Account and Register Customer

- «includes» relationships between transaction use cases and Update Balance

2.2. Class Diagram

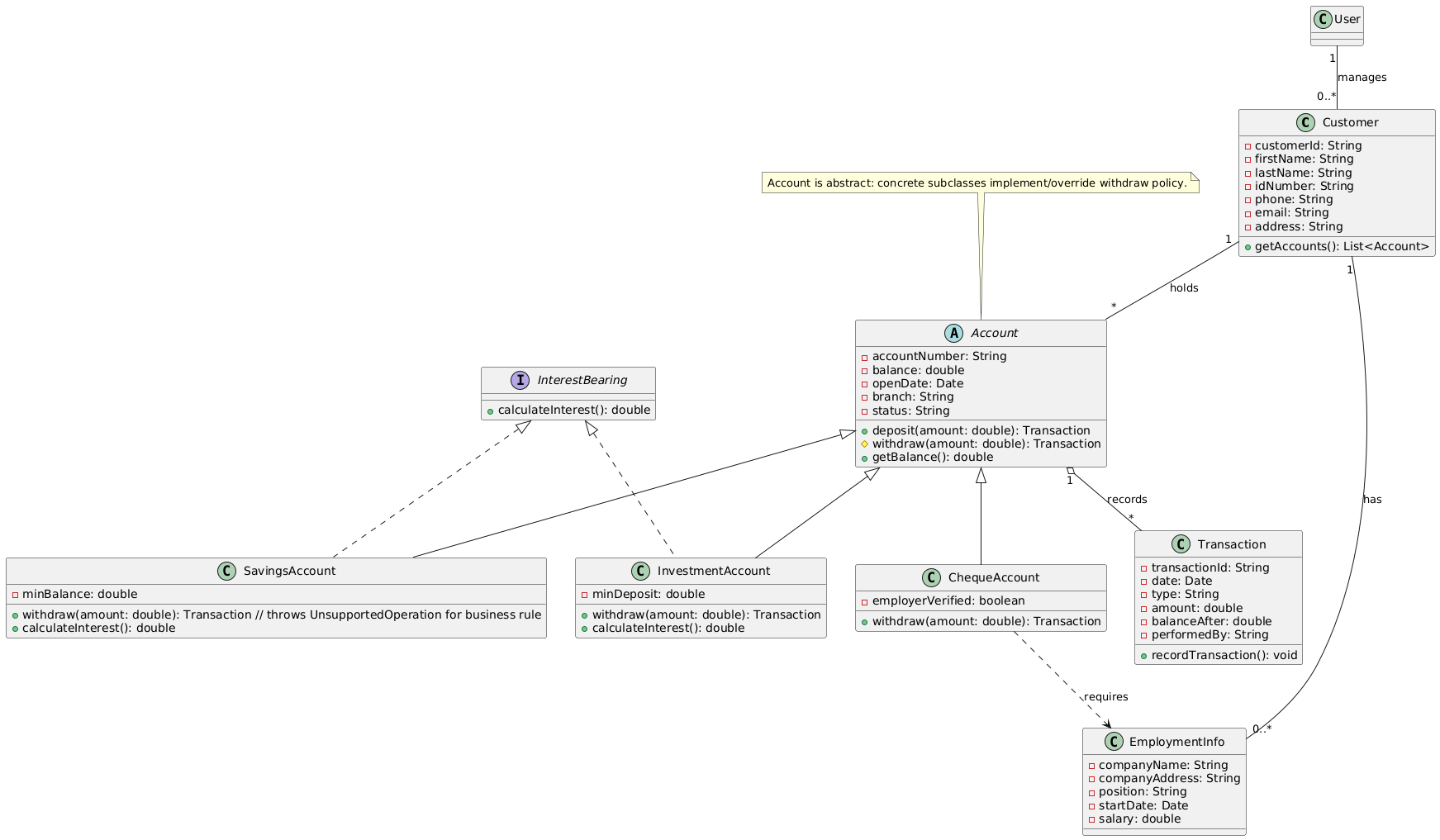


Figure 2: Banking System Class Diagram

Key Elements:

1. Abstraction

- Account class is abstract, providing common interface

- InterestBearing interface defines contract for interest calculation

2. Inheritance

- SavingsAccount, InvestmentAccount, and ChequeAccount inherit from Account

- Specialized classes override methods as needed

3. Interface

- InterestBearing interface with calculateInterest() method

- Implemented by SavingsAccount and InvestmentAccount

4. Encapsulation

- All attributes private with public getters/setters

- Proper access modifiers for all class members

5. Class Structures

- Appropriate associations between classes

- Correct multiplicity (e.g., Customer to Account: 1 to \*)

6. Polymorphism

- Method overriding in subclasses

- Interface implementation allowing polymorphic behavior

WEEK Sep. 15 – 19

3. Behavioural UML Modelling

3.1. Sequence Diagrams

Sequence Diagram 1: Deposit Funds

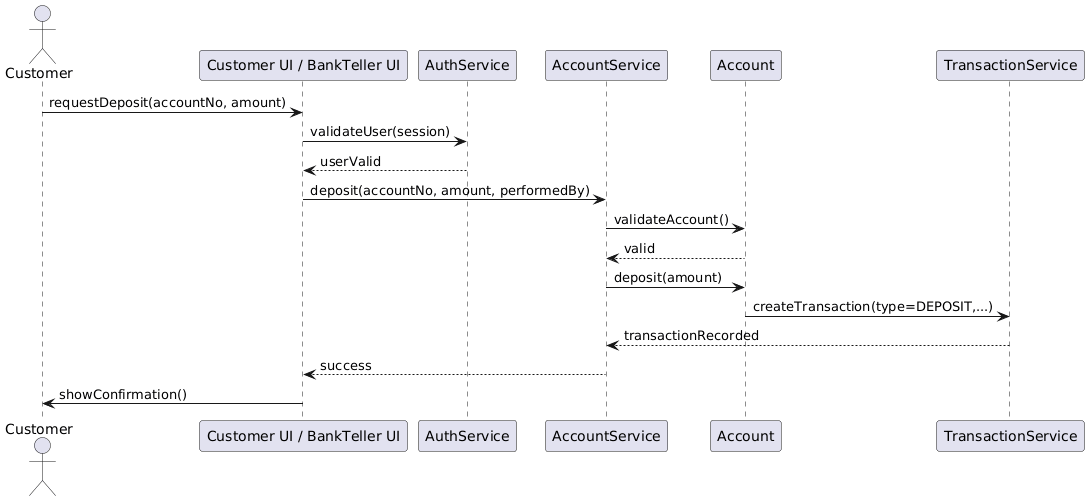


Figure 3: Sequence Diagram for Deposit Funds Use Case

Flow:

1. Customer initiates deposit request

2. System validates customer authentication

3. System retrieves account information

4. Customer provides deposit amount

5. System validates amount

6. System updates account balance

7. System creates transaction record

8. System confirms successful deposit

Sequence Diagram 2: Open Investment Account

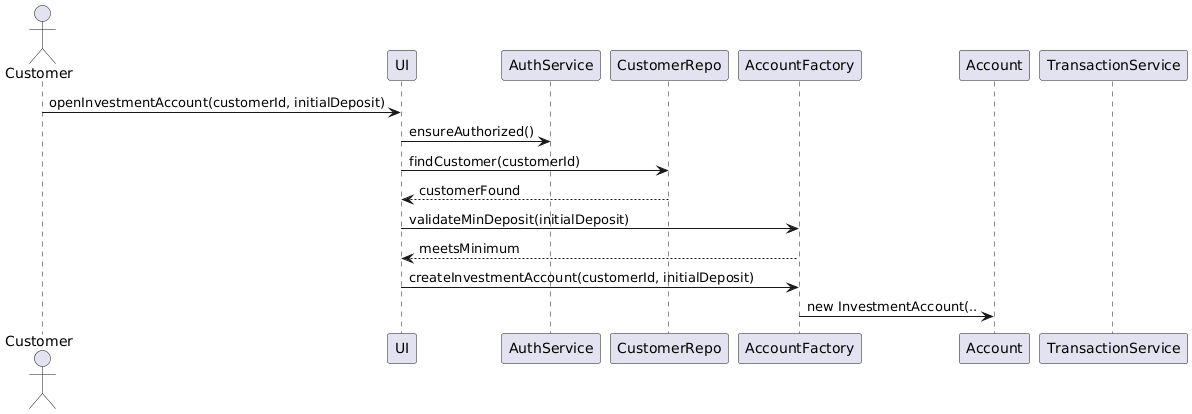


Figure 4: Sequence Diagram for Open Investment Account Use Case

Flow:

1. Customer requests to open Investment account

2. System verifies customer exists

3. System checks initial deposit meets minimum requirement

4. Customer provides initial deposit

5. System validates deposit amount

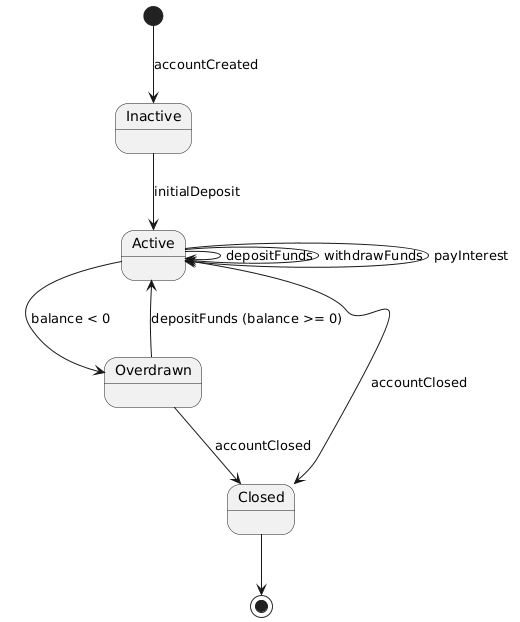
6. System creates new InvestmentAccount instance

7. System processes initial deposit transaction

8. System confirms account creation

3.2. State Diagram

State Diagram: PayInterest for InvestmentAccount



\*Figure 5: State Diagram for PayInterest Operation\*

States:

The diagram correctly models the account lifecycle.

* **Inactive**: new account awaiting first deposit.
* **Active**: normal state, supports deposits, withdrawals, and interest payments.
* **Overdrawn**: triggered when balance < 0; returns to Active once sufficient funds are deposited.
* **Closed**: final state where no transactions are possible.

The states and transitions accurately reflect banking rules and payment behaviour.